13.4 LOG BASED RECOVERY

(GGSIPU 2008, 2012, 2013; MDU May, 2009, 2010)

In this type of recovery technique, a log file is maintained which is a sequence of log records. Any operation which is performed on the database is recorded on the log. Separate logs are maintained for each and every activity of the database. Various types of log records are:

1. **$< T_i$ start $>$** It contains information about when a transaction T_i started.
2. **$< T_i, X_j, V_1, V_2 >$** It contains information that when transaction T_i writes data to item X_j. It also stores the information that V_1 is the old value of X_j and V_2 is the new value of X_j.
2. **$< T_i$ commit $>$** It contains information about when a transaction T_i has completed.
3. **$< T_i$ abort $>$** It contains information when a transaction is aborted due to any reason. This log always reside on stable storage. If any failure occurs in between the transaction, database can be restored to original state prior to failure by reading log record and undoing the transaction.

Whenever any failure occurs during a transaction two possibility arises:

1. **Database has been damaged:** In this case, the last backup copy of the database has to be restored and then all operations performed on the database since the last backup copy have to be reapplied by reading the contents of log file.
2. **Database has become inconsistent:** In this case it is enough to undo all operation that were performed during a transaction so that the database is known in a state prior to the start of that transaction.

In case of concurrent execution, whenever a crash occurs, the database would contain action of both committed and uncommitted transaction. In this case, redo log will first restore the database to a state prior to crash. Then undo log will be used to rollback the actions of uncommitted transaction.

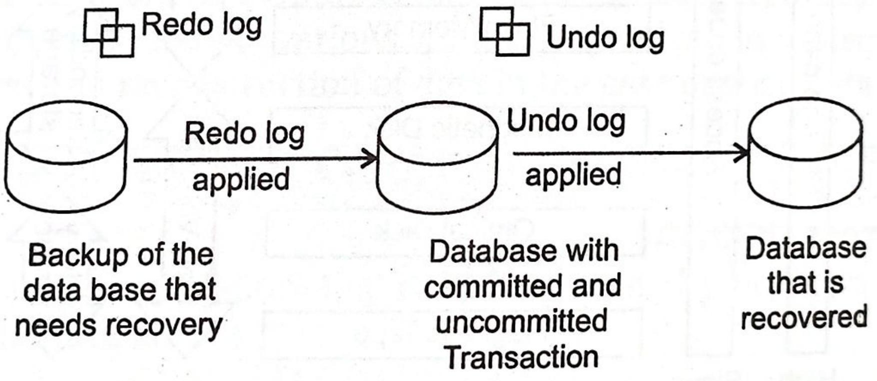
Redo log
Undo log
Redo log applied
Undo log applied
Backup of the data base that needs recovery
Database with committed and uncommitted Transaction
Database that is recovered

Fig. 13.2: Recovery Process

Various techniques of Database recovery are:

13.4.1 Recovery based on Deferred Update

(GGSIPU 2013, UPTU 2011-12)

Any updates to be applied by a transaction are deferred (postponed) until the transaction reaches its commit point.

A transaction reaches to its commit point only when all its update operation are recorded in the log and written to disk. Whenever a transaction T_1 executes a log is written to disk, $< T_1$, start $>$ is written to log whenever a transaction starts. If the transaction T_1 executes a write operation on data item X then its new value is written to the log $< T_1$, X, New value $>$. When all write operation are written to the log the transaction commits and this information is also entered into the log $< T_1$, commit $>$. Only new value is written to log, because database already contains old value in case of deferred update.

Example: Consider two transactions T_1 and T_2 in which Rs.500 is to be transferred from account A to B by T_1 and Rs.100 is withdrawn from account by T_2.

T_1	T_2
read (A) $A = A - 500$ write (A) read (B) $B = B + 500$ write (B)	read (C) $C = C - 100$ write (C)

Fig. 13.3: Transactions T_1 and T_2

Assume T_1 and T_2 executes serially and values of A, B, C before execution are 5000, 2000 and 1000 respectively.

➲ After executing T_1 and T_2 log is as follows:

Log	Database
$< T_1$, start $>$	
$< T_1$, A, 4500 $>$	
$< T_1$, B, 2500 $>$	
$< T_1$, commit $>$	

	A = 4500
	B = 2500

$< T_2$, start $>$	
$< T_2$, C, 900 $>$	
$< T_2$, commit $>$	

Fig. 13.4: Log Record for Transaction T_1 and T_2.

Value of A and B changes only when T_1 commits and is written to log.

➲ Now, suppose a failure occurs just after write (B), then log would be as follows:

$< T_1$, start $>$
$< T_1$, A, 4500 $>$
$< T_1$, B, 2500 $>$

Since there is no commit record, ($< T_1$, commit $>$) it means values of A and B need not be changed, so values of A will be 5000 and B will be 2000.

➲ Now, let failure occurs after write (C)

Then, log would be
$< T_1$, start $>

$< T_1, A, 4500 >$
$< T_1, B, 2500 >$
$< T_1$, commit $>$
$< T_2$, start $>$
$< T_2, C, 900 >$

Since, T_1 has committed, it is required to change values of A and B to 4500 and 2500 respectively, so, Redo (T_1) is performed. The value of C will remain 1000 because T_2 was not committed before crash took place.

13.4.2 Recovery Based on Immediate Update

Any update performed by the transaction on the database are applied immediately, without need to wait for transaction to reach its commit point. However, an update operation should still be recorded in log before applying to the database. Whenever a write operation take place it is written to the log in the following format:

$< T_i, X_j, V_1, V_2 >$

Where

T_i = ith transaction
X_j = Data item to be modified
V_1 = add value
V_2 = new value

Both undo and redo operation can be applied in this.

Example: Consider the transaction T_1 and T_2 in which Rs.500 is to be transferred from account A to B by T_1 and Rs.100 is to be withdrawn from account C by T_2.

T_1	T_2
read (A)	read (C)
A = A-500	C = C-100
write (A)	write (C)
read (B)	
B = B + 500	
write (B)	

Fig. 13.5: Transaction T1 and T2

Assume T_1 and T_2 executes serially and values of A, B, C before execution are 5000, 2000 and 1000 respectively. The content of log and database are:

Log	Database
$< T_1$, start $>$	
$< T_1$, A, 5000, 4500 $>$	
	A = 4500
$< T_1$, B, 2000, 2500 $>$	
	B = 2500
$< T_1$, commit $>$	
$< T_2$, start $>$	
$< T_2$, C, 1000, 900 $>$	
	C = 900
$< T_2$, commit $>$	

Fig. 13.6: Log Record for Transaction T1 and T2

Suppose a failure occurred

(a) Just after write (B)

Log Record are

$< T_1$, start $>$
$< T_1$, A, 5000, 4500 $>$
$< T_1$, B, 2000, 2500 $>$

Since there is $< T_1$, start $>$ and no $< T_1$, commit $>$. This means transaction T_1 must be undone and then T_1 and T_2 must be re-executed.

(b) Just after write (C)

Log records are

$< T_1$, start $>$
$< T_1$, A, 5000, 4500 $>$
$< T_1$, B, 2000, 2500 $>$
$< T_1$, commit $>$
$< T_2$, start $>$.
$< T_2$, C, 1000, 900 $>$

Here, there is no $< T_2$, commit $>$ corresponding to $< T_2$ start $>$. So, T_2 need to be undone. Also since $< T_1$, start $>$ and $< T_1$, commit $>$ both are present in the log so T_1 must be redone.

(c) After T_2 commit

Both T_1 and T_2 needs to be redone, since both $< T_1$, start $>$, $< T_1$, commit $>$ and $< T_2$, start $>$, $< T_2$, commit $>$ are present in the log.

13.6 CHECKPOINTS

(GGSIPU 2009, 2012)

The point of synchronisation between the database and the transaction log file is called a checkpoint. When a log file is used to recover from a failure, we may not know how far back in the log to search i.e. till where the database has successfully written to database. If such points are not known we may end up doing entire transaction again. So, a check point is used, to mark a point till where all database update have been done successfully. During a failure the most recent checkpoint is searched. All transaction after this checkpoint needs to be redone. Advantages of checkpoints are that there is no need to redo complete transaction before a checkpoint.

Following operations are performed when a checkpoint is implemented.

- All log records in main memory are output to stable storage.
- All buffer blocks are output to disk.
- A log record <checkpoint> is written to stable storage